REMARKS

In response to the Office Action dated May 27, 2008, Applicants respectfully request reconsideration and withdrawal of the rejections of the claims.

Claims 1-3, 6, 7, 29 and 31 were rejected under 35 U.S.C. §103, on the grounds that they were considered to be unpatentable over the Java Remote Method Invocation Specification published February 10, 1997 by Sun Microsystems (identified in the Office Action as "JDOM"), in view of the Jones et al. patent (U.S. 6,557,032). Claims 4, 5, 8-11, 15, 17, 20, 22, 25, 27 and 30 were rejected under 35 U.S.C. §103, on the basis of the JDOM reference and Jones patent, in further view of prior art described in the present application (identified as "APA"). Claims 12 and 13 were rejected under 35 U.S.C. §103, on the basis of the JDOM reference, the Jones patent, and APA, in further view of the Dattatri patent (U.S. 6,658,453). The remaining claims were rejected under 35 U.S.C. §103 on the basis of the Jones patent in view of APA. For the reasons set forth hereinafter, it is respectfully submitted that these references fail to suggest the claimed subject matter to a person of ordinary skill in the art, whether they are considered individually or in the combinations set forth in the Office Action.

In rejecting claim 1, the Office Action asserts that all of the features of this claim are disclosed in the JDOM reference, with the exception of the server machine residing in a smart device, and the client machine accessing the smart device via a reader. However, it is respectfully submitted that there are other, fundamental differences between the subject matter of claim 1 and the disclosure of JDOM. These differences can be understood with reference to Figures 4 and 5 of the present application. Figure 4 illustrates the event timing diagram for a remote

procedure call, as applied to a client/server system, for example a smart card and a terminal. In the client machine, e.g., the terminal, a client application 401 invokes a remote method, by sending a local call to a client stub 403. The client stub functions to marshal parameters that are used by the remote method, and sends them to the server by means of respective communication modules 407 and 409. At the server, the remote procedure call request is dispatched to the appropriate server stub 413, which unmarshals the parameters, and invokes the requested method from an applet 415, or the like. The result of the execution of the method by the applet is returned to the client through the auspices of the server stub 413 and the client stub 403.

As discussed in the background portion of the present application, this architecture imposes certain constraints on program developers, and forces them to design a proprietary protocol for every new application. In particular, it requires the designer to focus upon low-level byte string structures and protocols for each application. The claimed subject matter relieves the designer of such constraints, and enables them to concentrate on the overall application design, rather than specific low level structures.

Referring to Figure 5, this capability is accomplished through the use of direct method invocation (DMI) and an interface that specifies the methods provided by a card applet, which are visible to the applet's clients. The interface is used to construct a client stub, or proxy, for the client, and to provide a description of the applet to the run-time environment of the server. In operation, the client application 501 invokes a method of interest through the proxy 503 that was constructed from the applet's interface. The proxy prepares a DMI invocation message with parameter values provided by the client application. This message is sent to the

server, and at the server it is received by the run-time environment 513. The run-time environment unmarshals the parameters and invokes the requested method of the applet 515. The results from the execution of the method are marshaled by the run-time environment, and returned to the client as a DMI reply message.

Claim 1 recites a method that includes the steps of generating a local object at the client machine that operates as a proxy to a remote object that is resident on a server machine, and referencing the local object by an application at the client machine, to cause the local object to marshal parameters and send a process level call request to the server machine. Claim 1 goes on to recite that "responsive to said request by the server machines run-time environment, said run-time environment causing the parameters in the request to become unmarshalled, said remote object to be executed ..." Thus, claim 1 recites that the server's *run-time environment* receives the request sent from the client machine, and that *this run-time environment* causes the parameters in the request to become unmarshalled, and the remote object to be executed.

In connection with this particular claimed feature, the Office Action states " 'server machine's run-time environment [...] causing the parameters in the request to become unmarshalled' as a *skeleton* for a remote object..." (emphasis added). It is respectfully submitted that the skeleton described on pages 17 and 18 of the JDOM reference is not the same as, nor does it correspond to, a run-time environment in a server. A run-time environment is part of a computer's operating system, or software that runs on top of the operating system. In other words, it is closely associated with the basic operations of the computer. In contrast, a stub is a small piece of code that is used to represent some other programming functionality,

and typically used in distributed programming. It does not provide the same functionality, nor possess the same characteristics, as a run-time environment.

The stub/skeleton architecture described in chapter 3 of the JDOM reference, and illustrated on page 17 thereof, more closely corresponds to the architecture illustrated in Figure 4 of the present application, in which respective stubs are located on the client machine and the server machine, and exist independently of the operating systems for those machines. The JDOM reference does not disclose, nor otherwise suggest, an architecture that operates in the manner set forth in claim 1, wherein a process level call request is received, and its parameters unmarshalled, by the run-time environment of the server. In other words, it does not disclose the direct method invocation that is encompassed by the claim.

For at least these reasons, therefore, it is respectfully submitted that the JDOM reference does not suggest the subject matter of claim 1, whether or not it is considered in combination with the Jones patent or any of the other cited references. For at least these same reasons, it is respectfully submitted that claims 2-9 and 29-31 are likewise patentable over these references.

As noted on page 14 of the present application, an advantage of the DMI protocol is that it has only one command. As such, a method can be invoked, and a response returned, using a single APDU command/response pair. Claims 14-28 encompass this aspect of the disclosed invention. For example, claim 14 recites an applet proxy that is configured to generate a single command APDU in response to a local call, and a run-time environment that is configured to generate the local call on a smart device to invoke the method "in response to the single command APDU without the applet having been selected with another command APDU."

In rejecting claim 14, the Office Action acknowledges that the Jones patent does not disclose an applet proxy that is configured to generate a single command APDU, but contends that this subject matter is taught by APA. It is respectfully submitted, however, that APA does not suggest the subject matter of claim 14. At page 10, the present application describes the known procedure, in which, prior to sending a command to an applet, it is necessary to send an initial APDU that causes the applet to be selected.

In contrast to this disclosed subject matter, claim 14 recites that the run-time environment invokes the method in response to the single command APDU "without the applet having been selected with another command APDU." The rejection of claim 14 does not address this quoted feature of the claim. The fact that an applet proxy is capable of exchanging APDUs, per se, does not suggest a run-time environment that invokes a method in response to a single command APDU, without a prior APDU that selects the applet that provides the method.

For at least this reason, therefore, it is respectfully submitted that the subject matter of claim 14 is not suggested by the Jones patent, even when it is considered in light of the prior art described in Applicant's specification. For at least this same reason it is respectfully submitted that claims 15-28 are patentably distinct from the cited prior art.

In view of the foregoing, it is respectfully submitted that all pending claims are patentable over the prior art of record. Reconsideration and withdrawal of the rejections, and allowance of all claims is respectfully requested.

Respectfully submitted,

BUCHANAN INGERSOLL & ROONEY PC

Date: November 26, 2008 By: /James A. LaBarre/
James A. LaBarre
Registration No. 28632

P.O. Box 1404
Alexandria, VA 22313-1404
703 836 6620